영상처리 Python 실습 과제 1

201620350 김지영

(1-1~1-3: 다음 python code를 이용하여 실험하고 결과 영상이 생성되는 이유를 설명할 것)

1-1. 이진화(thresholding)의 임계값을 바꾸면서 출력 영상의 변화를 관찰한다. (두 개의 color 영상에 대해서 실험할 것)

=> Thresholding 기법은 영상에서 각각의 픽셀들의 값을 특정한 임계값 보다 낮은 픽셀은 검정색(0)으로, 임계값과 같거나 높은 픽셀은 흰색(255)으로 바꾸는 방법입니다. Thresholding 기법을 적용하여 영상의 결과를 확인해보고 임계값을 조절하여 어떻게 달라지는지 알아보았습니다.

[그림 1]

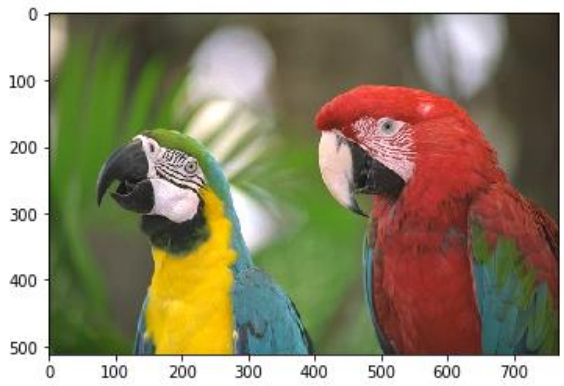
(1) Threshold=50 인 경우

- Python Code

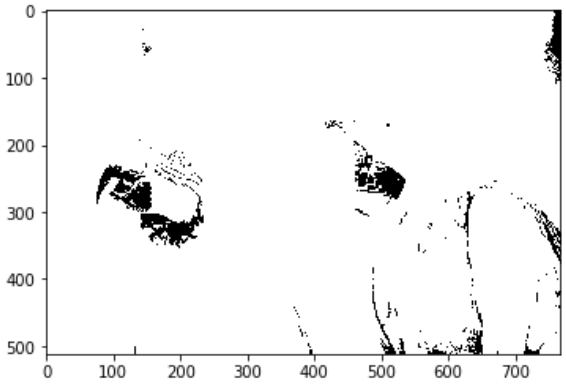
```
import scipy
import numpy as np
from scipy import ndimage
from matplotlib import pyplot as plt

img = plt.imread('parrot.png')
img = np.uint8(255*img)
plt.imshow(img)
plt.show()

#Image threshold
h,w,z = img.shape
lumimg = img[:,:,0] * 0.2126 + img[:,:,1] * 0.7152 + img[:,:,2] * 0.0722
lumimg = np.uint8(lumimg)
imgthres = np.zeros([h,w])
for i in range(h):
    for j in range(w):
```

```
        if(lumimg[i,j]>=50): # 임계값을 50으로 준 것
            imgthres[i,j]=255
        else:
            imgthres[i,j]=0
imgthrescolor = np.zeros(img.shape)
imgthrescolor[:,:,0] = imgthrescolor[:,:,1]= imgthrescolor[:,:,2] = imgthres[:,:]
plt.imshow(imgthrescolor)
plt.show()
```

- Image



Input Image



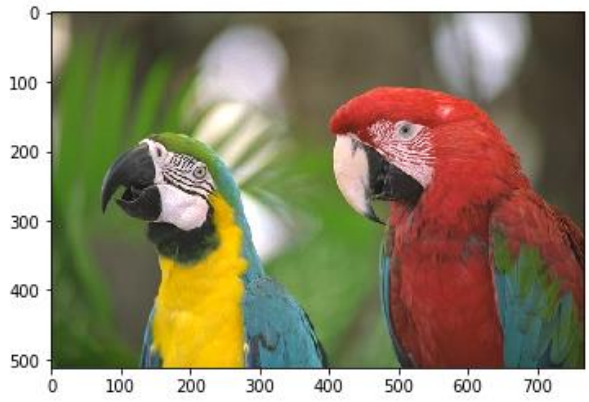
Output Image

(2) Threshold=128 인 경우

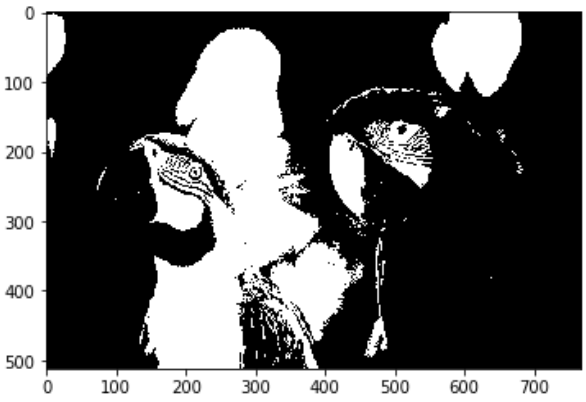
- Python Code

```
if(lumimg[i,j]>=128): # 위의 코드에서 threshold값을 128으로 바꿔주었습니다.
```

-Image



Input Image



Output Image

=> 임계값이 50일때가 128일 때 보다 더 흰부분이 많이생겨, 원본이미지에서 완전히 벗어난 모습을 확인할 수 있습니다. 임계값이 50일때는 128보다 0에 훨씬 가깝기 때문에 (임계값이 상대적으로 작기 때문에), 흰색으로 표현되는 픽셀수가 많아져서 이미지가 더 하얗게 보이며 원본이미지에서 확실하게 어두운 부분만 찾아냈습니다.

[그림 2]

(1) Threshold=50 인 경우

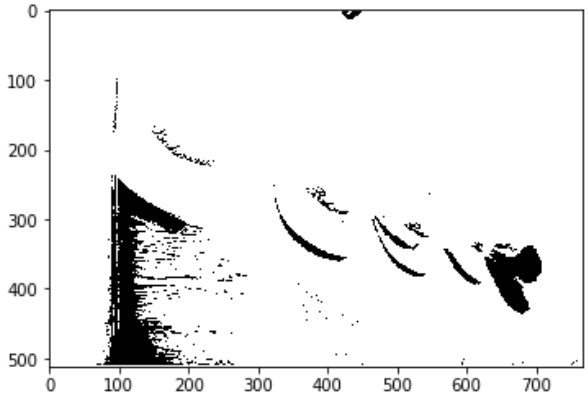
- Python Code

```
import scipy
import numpy as np
from scipy import ndimage
from matplotlib import pyplot as plt

img = plt.imread('cap.png')
img = np.uint8(255*img)
plt.imshow(img)
plt.show()

#Image threshold
h,w,z = img.shape
lumimg = img[:,:,0] * 0.2126 + img[:,:,1] * 0.7152 + img[:,:,2] * 0.0722
```

```
lumimg = np.uint8(lumimg)
imgthres = np.zeros([h,w])
for i in range(h):
    for j in range(w):
        if(lumimg[i,j]>=50): # 임계값을 50으로 준 것
            imgthres[i,j]=255
        else:
            imgthres[i,j]=0
imgthrescolor = np.zeros(img.shape)
imgthrescolor[:,:,0] = imgthrescolor[:,:,1]= imgthrescolor[:,:,2] = imgthres[:,:]
plt.imshow(imgthrescolor)
plt.show()
```

- Image

Input Image



Output Image

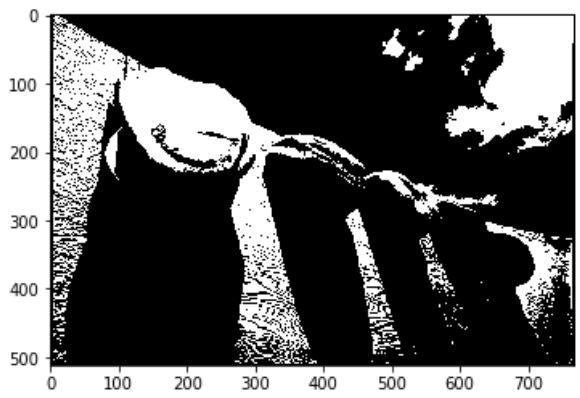
(2) Threshold=128 인 경우

- Python Code

```
if (lumimg[i,j]>=128): # 위의 코드에서 threshold값을 128으로 바꿔주었습니다.
```

- Image

Input Image



Output Image

=> [그림 1]과 마찬가지로, 임계값이 50일때가 128일 때 보다 더 흰부분이 많이생겨, 원본이미지에서 완전히 벗어난 모습을 확인할 수 있습니다. 임계값이 50일때는 128보다 0에 훨씬 가깝기 때문에 (임계값이 상대적으로 작기 때문에), 흰색으로 표현되는 픽셀수가 많아져서 이미지가 더 하얗게 보이며 원본이미지에서 확실하게 어두운 부분만 찾아냈습니다.

1-2. 감마 보정을 적용하여 입력과 결과 영상을 비교한다. (두 개의 color 영상에 대해서 실험할 것)

=> 감마 보정법은 감마의 역수를 최종 출력 이미지에 적용하는 기법입니다. 역 감마곡선을 선형 출력 색상에 곱해 빛의 강도를 비선형적으로 변형하는 방식으로 영상의 밝기를 조절할 수 있습니다. 출력하고자 하는 이미지를 $\boldsymbol{O} = \boldsymbol{I}^{\frac{1}{\gamma}}$ 이라고 정의한후, 감마 보정을 적용하여 영상의 결과를 확인해보고 감마 값에 따라 어떻게 달라지는지 알아보았습니다.

[그림 1]

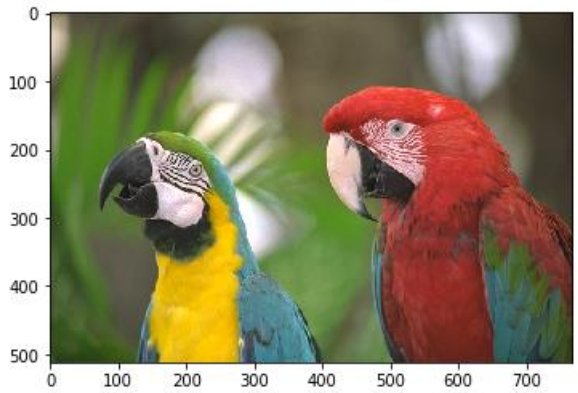
(1) gamma=2.2 인 경우

- Python Code

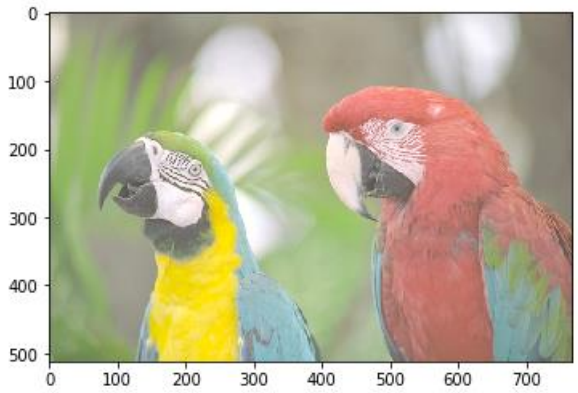
```
import scipy
import numpy as np
from scipy import ndimage
from matplotlib import pyplot as plt
```

```
img = plt.imread('parrot.png')
img = np.uint8(255*img)

plt.imshow(img)
plt.show()

#gamma correction
gcorimg = np.power(img/255.0,1/2.2)
gcorimg = gcorimg*255.0
gcorimg = np.uint8(gcorimg)
plt.imshow(gcorimg)
plt.show()
```

- Image



Input Image



Output Image

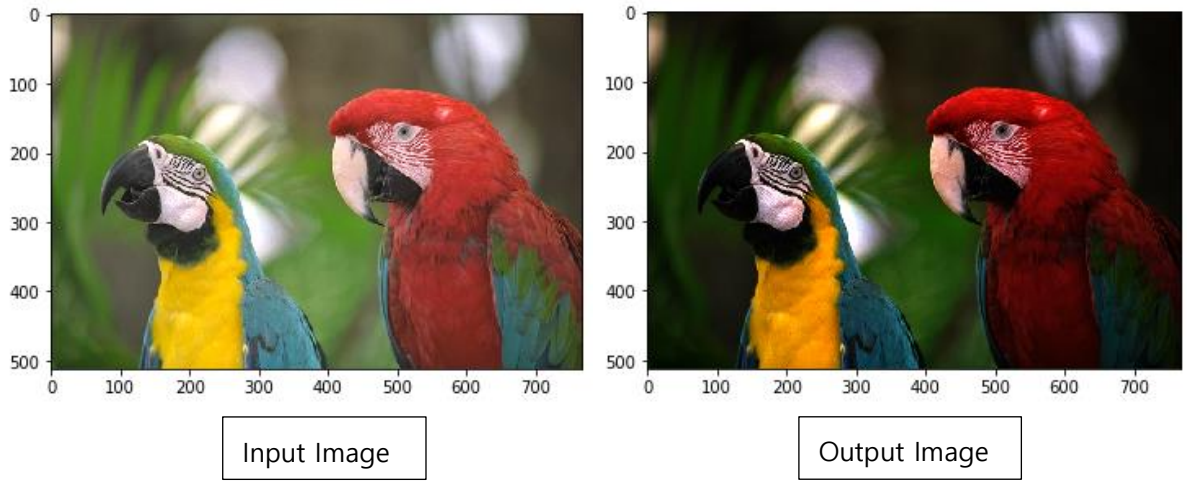
(2) gamma=1/2.2 인 경우

- Python Code

```
import scipy
import numpy as np
from scipy import ndimage
from matplotlib import pyplot as plt

img = plt.imread('parrot.png')
```

```
img = np.uint8(255*img)


plt.imshow(img)
plt.show()

#gamma correction
gcorimg = np.power(img/255.0,2.2)
gcorimg = gcorimg*255.0
gcorimg = np.uint8(gcorimg)
plt.imshow(gcorimg)
plt.show()
```

- Image



Input Image

Output Image

=> gamma값이 1보다 큰 2.2를 입력했을 때, 결과 이미지가 원본 이미지보다 더 밝아졌으며, gamma값이 1보다 작은 1/2.2를 입력했을 때는 결과 이미지가 원본 이미지보다 더 어두워졌음을 확인할 수 있습니다. 이렇게 감마 보정법을 통해 이미지 밝기의 균형을 유지시킬 수 있습니다.

[그림 2]

(1) gamma=2.2 인 경우

- Python Code

```
import scipy
import numpy as np
from scipy import ndimage
from matplotlib import pyplot as plt

img = plt.imread('parrot.png')
img = np.uint8(255*img)

plt.imshow(img)
plt.show()

#gamma correction
gcorimg = np.power(img/255.0,1/2.2)
gcorimg = gcorimg*255.0
gcorimg = np.uint8(gcorimg)
plt.imshow(gcorimg)
plt.show()
```

- Image

Input Image

Output Image

(2) gamma=1/2.2 인 경우

- Python Code

```
import scipy
import numpy as np
from scipy import ndimage
from matplotlib import pyplot as plt

img = plt.imread('parrot.png')
img = np.uint8(255*img)

plt.imshow(img)
plt.show()

#gamma correction
gcorimg = np.power(img/255.0,2.2)
gcorimg = gcorimg*255.0
gcorimg = np.uint8(gcorimg)
plt.imshow(gcorimg)
plt.show()
```

- Image

Input Image

Output Image

=> [그림 1]과 마찬가지로, gamma값이 1보다 큰 2.2를 입력했을 때, 결과 이미지가 원본 이미지보다 더 밝아졌으며, gamma값이 1보다 작은 1/2.2를 입력했을 때는 결과 이미지가 원본 이미지

보다 더 어두워졌음을 확인할 수 있습니다. 이렇게 감마 보정법을 통해 이미지 밝기의 균형을 유지시킬 수 있습니다.

1-3. 평균 필터의 마스크 크기를 바꾸면서 필터링하고 결과를 관찰한다. (parrot.png와 lena_512_noise.png 영상에 대해서 실험할 것, lena_512_noise.png 영상에 대해서는 흑백 영상에 맞도록 code 수정)

=> 공간상의 평균 필터링이란, 영상 신호에 대하여 공간 영역(Spatial Domain)에서의 필터 처리를 의미합니다. 하나의 픽셀 값을 구하기 위해 주변 픽셀 값들을 이용하는 방식으로 필터링 되는데, 이때 중앙에 처리 값이 저장되므로 반드시 마스크는 홀수X홀수의 형태를 가져야 합니다. 필터링 한 영상의 결과를 확인해보고 필터의 마스크 크기에 따라 결과 값이 어떻게 달라지는지 알아보았습니다.

[그림1 - 'parrot.png']

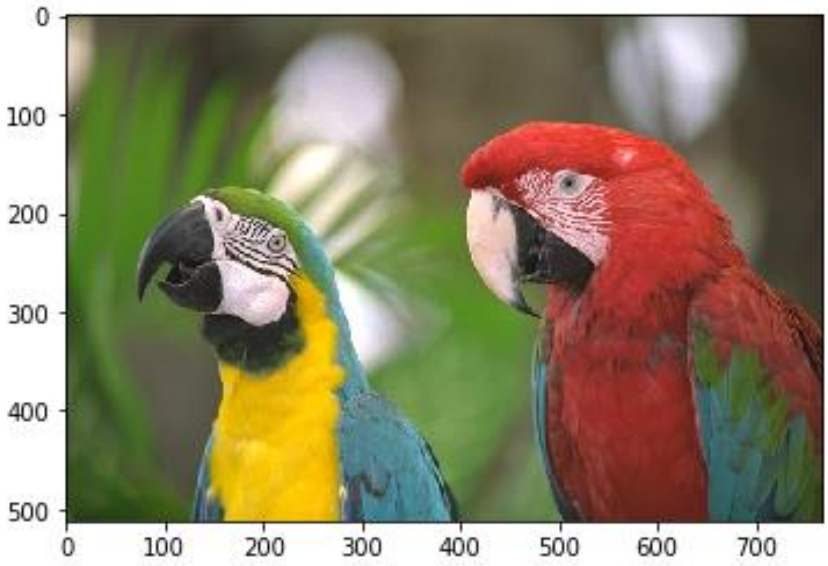
(1) 3x3 마스크 필터인 경우

- Python Code

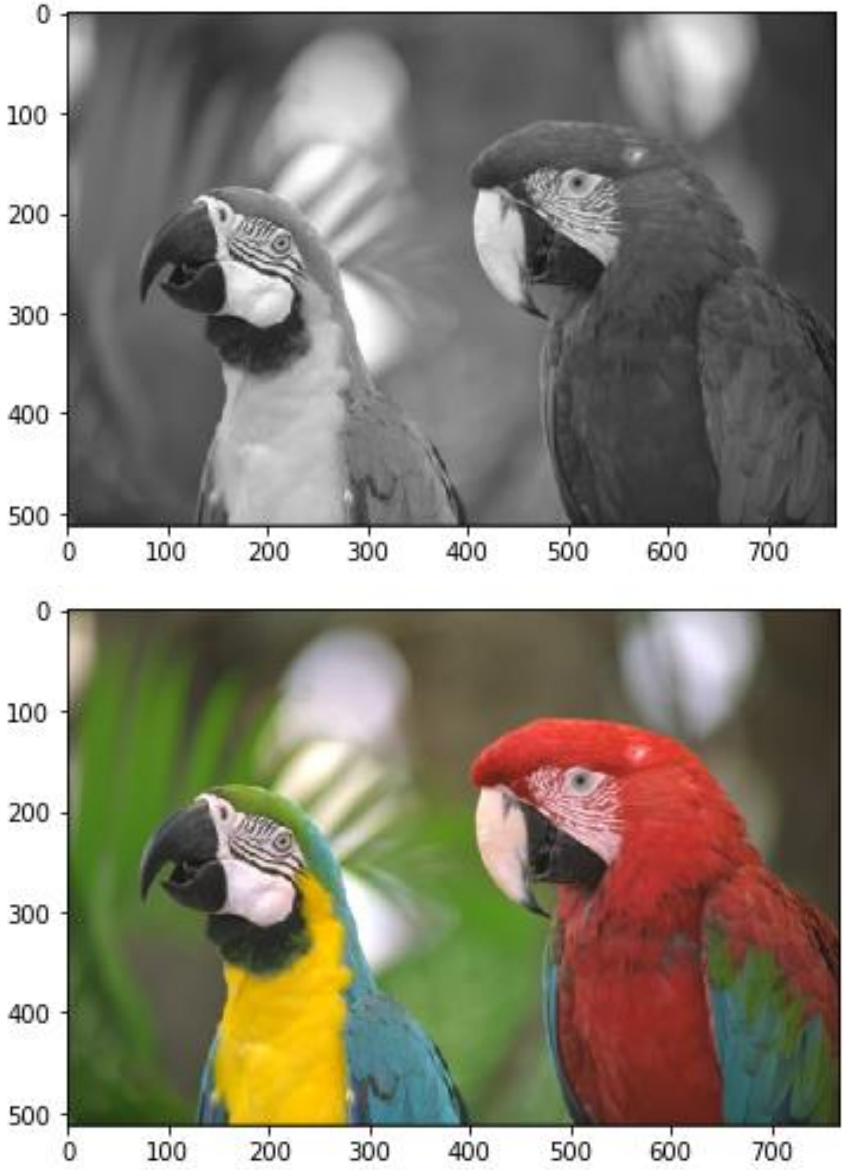
```
import scipy
import numpy as np
from scipy import ndimage
from matplotlib import pyplot as plt

kernel1 = np.array([[1/9, 1/9, 1/9],
                    [1/9, 1/9, 1/9],
                    [1/9, 1/9, 1/9]])

img = plt.imread('parrot.png')
img = np.uint8(255*img)

plt.imshow(img)
plt.show()
h,w,z = img.shape
```

```
lumimg = img[:,:,0] * 0.2126 + img[:,:,1] * 0.7152 + img[:,:,2] * 0.0722


lowpass = ndimage.convolve(lumimg, kernel1)
lowpassc= np.zeros(img.shape)
lowpassc[:,:,0] = lowpassc[:,:,1]= lowpassc[:,:,2] = lowpass[:,:]
lowpassc=np.uint8(lowpassc)
plt.imshow(lowpassc)
plt.show()

lowpass1= np.zeros(img.shape)
lowpass1[:,:,0] = ndimage.convolve(np.float32(img[:,:,0]), kernel1)
lowpass1[:,:,1] = ndimage.convolve(np.float32(img[:,:,1]), kernel1)
lowpass1[:,:,2] = ndimage.convolve(np.float32(img[:,:,2]), kernel1)
lowpass1=np.uint8(lowpass1)
plt.imshow(lowpass1)
plt.show()
```

- Image



Input Image



Output Images

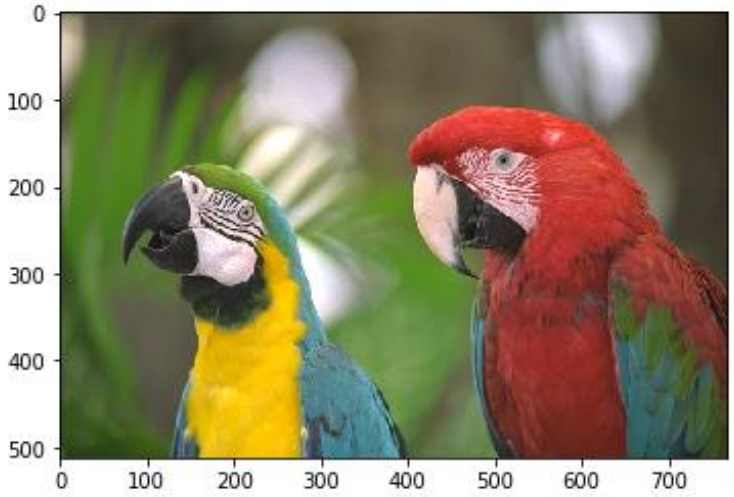
(2) 7x7 마스크 필터인 경우

- Python Code

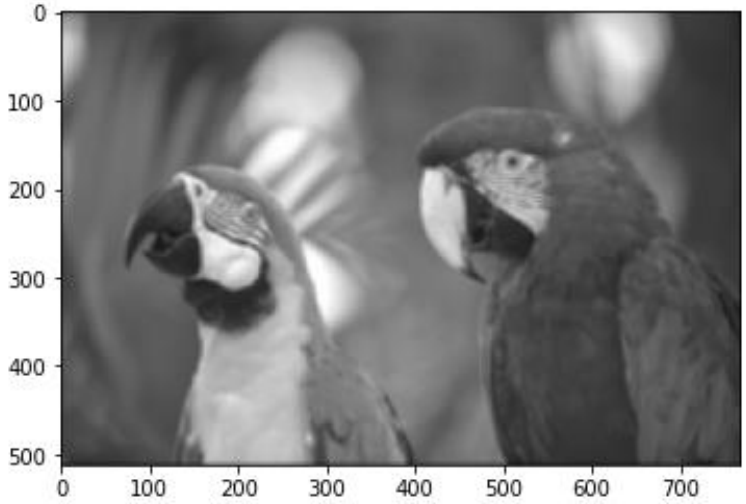
```
import scipy
import numpy as np
from scipy import ndimage
from matplotlib import pyplot as plt

kernel2 = np.array([[1/49, 1/49, 1/49, 1/49, 1/49, 1/49, 1/49],
```

```
                            [1/49, 1/49, 1/49, 1/49, 1/49, 1/49, 1/49],
                            [1/49, 1/49, 1/49, 1/49, 1/49, 1/49, 1/49],
                            [1/49, 1/49, 1/49, 1/49, 1/49, 1/49, 1/49],
                            [1/49, 1/49, 1/49, 1/49, 1/49, 1/49, 1/49],
                            [1/49, 1/49, 1/49, 1/49, 1/49, 1/49, 1/49],
                            [1/49, 1/49, 1/49, 1/49, 1/49, 1/49, 1/49]])

img = plt.imread('parrot.png')
img = np.uint8(255*img)

plt.imshow(img)
plt.show()

h,w,z = img.shape
lumimg = img[:,:,0] * 0.2126 + img[:,:,1] * 0.7152 + img[:,:,2] * 0.0722

lowpass = ndimage.convolve(lumimg, kernel2)
lowpassc= np.zeros(img.shape)
lowpassc[:,:,0] = lowpassc[:,:,1]= lowpassc[:,:,2] = lowpass[:,:]
lowpassc=np.uint8(lowpassc)
plt.imshow(lowpassc)
plt.show()

lowpass1= np.zeros(img.shape)
lowpass1[:,:,0] = ndimage.convolve(np.float32(img[:,:,0]), kernel2)
lowpass1[:,:,1] = ndimage.convolve(np.float32(img[:,:,1]), kernel2)
lowpass1[:,:,2] = ndimage.convolve(np.float32(img[:,:,2]), kernel2)
lowpass1=np.uint8(lowpass1)
plt.imshow(lowpass1)
plt.show()
```

- Image



Input Image



Output Images

=> 저주파 통과 필터링은 신호 성분 중에서 저주파 성분은 통과시키고 고주파 성분은 차단하는 필터입니다. 주로 잡음을 제거하거나, 흐릿한 영상을 얻을 때 주로 사용됩니다. 위의 이미지는 color 이미지에 3*3 필터와 7*7 필터를 통과시킨 모습입니다. 결과값을 비교해보면, 7*7 크기의 마스크로 필터링한 이미지가 3*3 일때보다 이미지 blurring이 더 심합니다 (더 흐릿해 보입니다). 이로써, 필터의 마스크 크기가 더 커질수록 영상이 더 흐릿해짐을 알 수 있습니다.

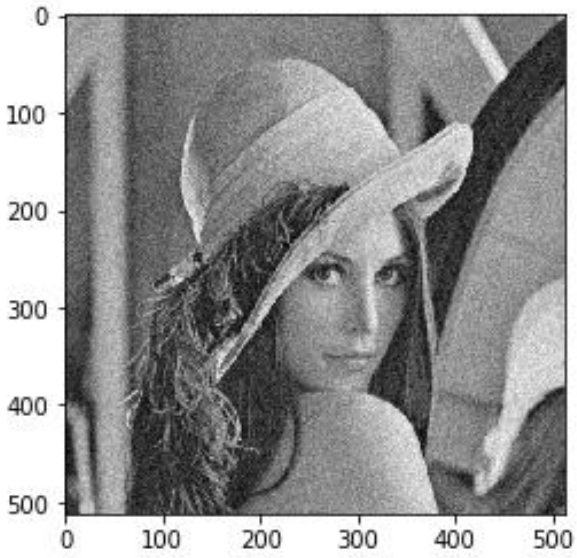
[그림2 - 'lena_512_noise.png']

(1) 3x3 마스크 필터인 경우

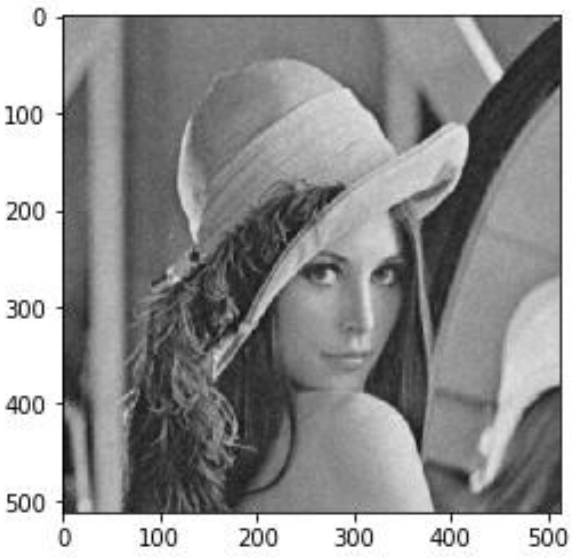
- Python Code

```
import scipy
import numpy as np
from scipy import ndimage
from matplotlib import pyplot as plt

inimg = plt.imread('lena_512_noise.png')
inimg = np.uint8(255*inimg)
plt.imshow(inimg,'gray',vmin=0,vmax=255) # gray 이미지로 변환한것
plt.show()

# 3*3 lowpass filter
kernel1 = np.array([[1/9, 1/9, 1/9],
                    [1/9, 1/9, 1/9],
                    [1/9, 1/9, 1/9]])

h,w = inimg.shape
lowpass1= np.zeros(inimg.shape)
lowpass1[:,:] = ndimage.convolve(np.float32(inimg[:,:]), kernel1)
lowpass1=np.uint8(lowpass1)
plt.imshow(lowpass1,'gray',vmin=0,vmax=255) # gray 이미지로 변환한것
plt.show()
```

- Image



Input Image



Output Image

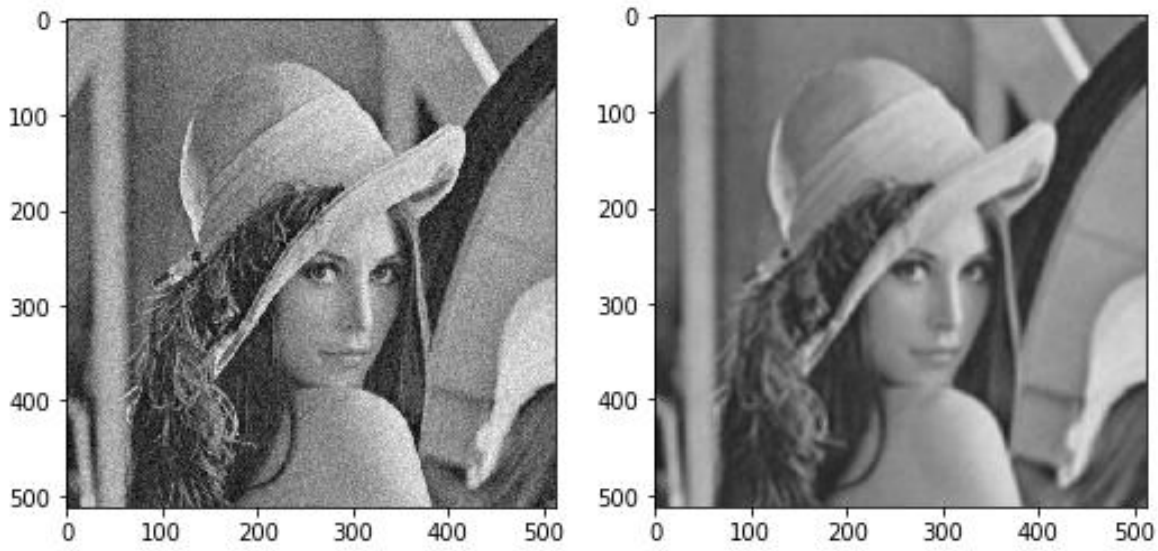
(2) 7x7 마스크 필터인 경우

- Python Code

```
import scipy
import numpy as np
from scipy import ndimage
from matplotlib import pyplot as plt

inimg = plt.imread('lena_512_noise.png')
inimg = np.uint8(255*inimg)
plt.imshow(inimg,'gray',vmin=0,vmax=255) # gray 이미지로 변환한것
plt.show()

# 7*7 lowpass filter
kernel2 = np.array([[1/49, 1/49, 1/49, 1/49, 1/49, 1/49, 1/49],
                    [1/49, 1/49, 1/49, 1/49, 1/49, 1/49, 1/49],
                    [1/49, 1/49, 1/49, 1/49, 1/49, 1/49, 1/49],
                    [1/49, 1/49, 1/49, 1/49, 1/49, 1/49, 1/49],
```

```
                    [1/49, 1/49, 1/49, 1/49, 1/49, 1/49, 1/49],
                    [1/49, 1/49, 1/49, 1/49, 1/49, 1/49, 1/49],
                    [1/49, 1/49, 1/49, 1/49, 1/49, 1/49, 1/49]])

h,w = inimg.shape
lowpass1= np.zeros(inimg.shape)
lowpass1[:,:] = ndimage.convolve(np.float32(inimg[:,:]), kernel2)
lowpass1=np.uint8(lowpass1)
plt.imshow(lowpass1,'gray',vmin=0,vmax=255) # gray 이미지로 변환한것
plt.show()
```

- Image



=> 위의 이미지는 [그림1]과 마찬가지로, 이미지에 3*3 필터와 7*7 필터를 통과시킨 모습입니다. 원본이미지를 흑백 이미지로 변환하기 위하여 코드를 바꾸었습니다. 결과값을 비교해보면, 7*7 크기의 마스크로 필터링한 이미지가 3*3 일때보다 이미지 blurring이 더 심합니다 (더 흐릿해 보입니다). 이로써, 필터의 마스크 크기가 더 커질수록 영상이 더 흐릿해짐을 알 수 있습니다.

또한, [그림2]는 noise가 있는 이미지인데, spatial average mask를 사용하면 random noise는 줄어들지만, blurring 현상은 생긴다는 것을 알 수 있습니다.

(2-1~2-3: Contrast stretching을 수행하고 결과 영상이 생성되는 이유를 설명할 것)

2-1. 다음과 같은 python program을 이용하여 contrast stretching 수행 (lena_512_low1.png, lena_512_low2.png, boats_512_high.png, lena.png 영상에 대해서 실험할 것)

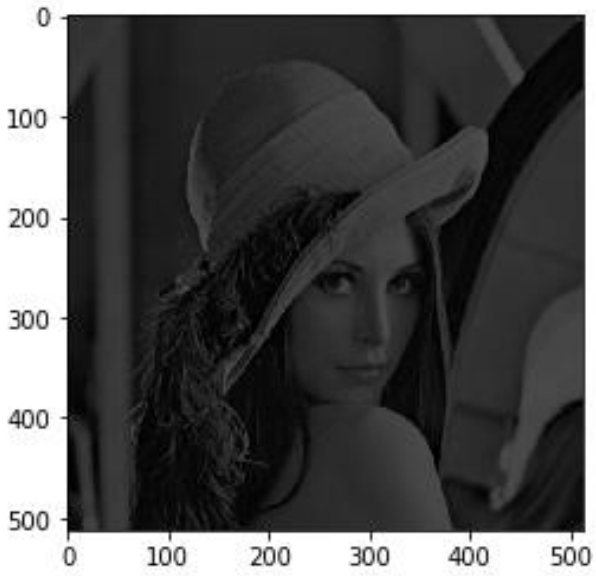
=> Contrast stretching는 명암비를 늘려 영상 화질을 향상시키는 기법입니다. Contrast stretching 기법을 적용하여 영상의 결과를 확인해보고, 결과 영상이 생성되는 이유를 알아보겠습니다.

[그림1 - 'lena_512_low1.png']

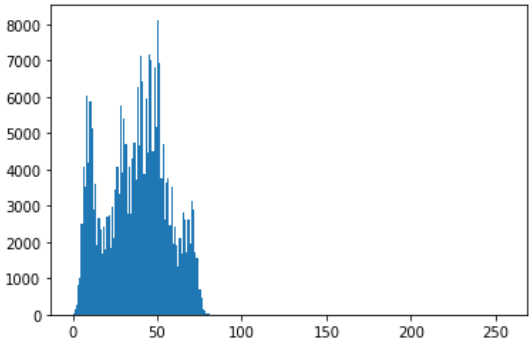
- Python Code

```
import scipy
import numpy as np
from scipy import ndimage
from matplotlib import pyplot as plt

inimg = plt.imread('lena_512_low1.png')
inimg = np.uint8(255*inimg)
plt.imshow(inimg,'gray',vmin=0,vmax=255)
plt.show()

#Contrast stretching
h,w = inimg.shape
maxval = np.amax(inimg)
minval = np.amin(inimg)
print(maxval,minval)

plt.hist(inimg.flatten(),256,[0,256])
plt.show()

outimg = np.zeros(inimg.shape)
outimg = 255*np.float32(inimg[:,:]-minval)/float(maxval-minval)
outimg = np.uint8(outimg)
plt.imshow(outimg,'gray',vmin=0,vmax=255)
plt.show()
```

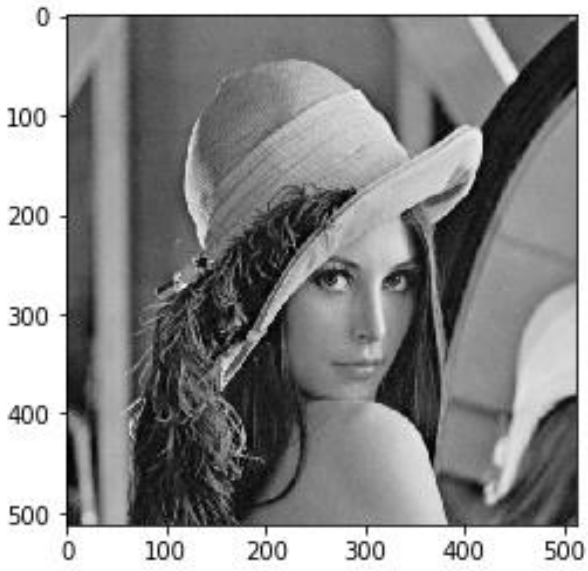
- Image



Input Image



histogram



Output Image

=> Stretching한 결과, 굉장히 어두웠던 입력 이미지에 비해 출력 이미지는 명암대비가 훨씬 높아졌습니다. 이를 구하기위해 이미지의 픽셀 값 중에서 밝기 값이 가장 작은 값을 `minval`, 가장 높은 값을 `maxval`이라고 코드에 입력해주었습니다. 그리고 그 값들의 차를 구한뒤에, `255`로 나누면 다음과 같이 표현할 수 있습니다.

```
outimg = 255*np.float32(inimg[:,:]-minval)/float(maxval-minval)
```

위 그림의 입력 이미지는 밝기 값이 작은 영역 즉, 어두운 영역에 영상 픽셀 값이 집중되어 있는데, 이것을 히스토그램 스트레칭 하여 집중되어 있는 것을 퍼트려서 영상의 가시도를 좋게 해준

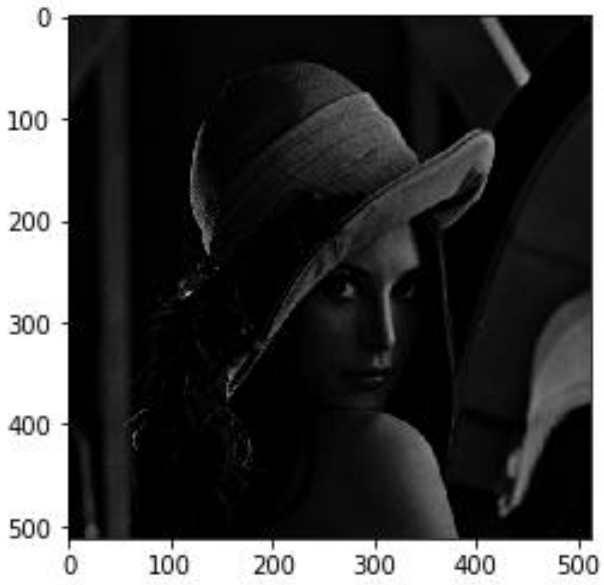
것입니다.

[그림2 - 'lena_512_low2.png']

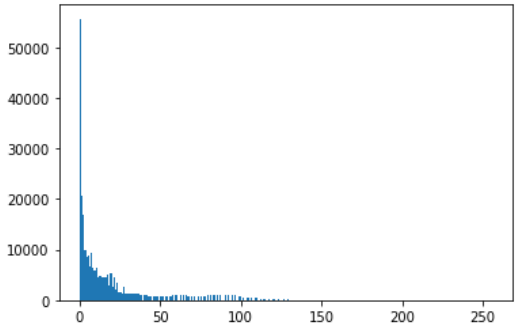
- Python Code

```
inimg = plt.imread('lena_512_low2.png') # 위의 코드 중, 이미지 파일명만 바꾼 것입니다.
```

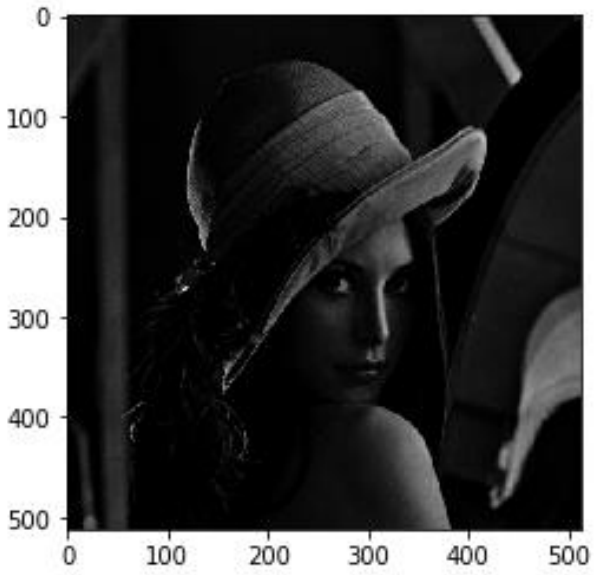
- Image



Input Image



histogram



Output Image

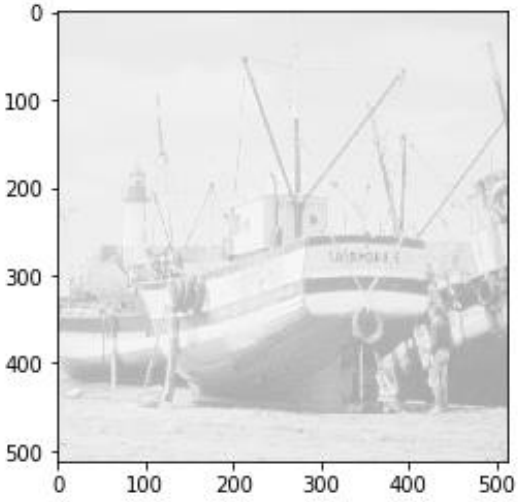
=> Stretching한 결과, 입력 이미지에 비해 출력 이미지의 명암대비가 높아지긴 했지만, 큰 차이가 없습니다. 히스토그램을 보면, [그림1]에서 보다 스트레칭이 되지 않아, 집중된 픽셀 값들이 퍼지지 못한 모습이지만, 기존의 흐릿했던 느낌의 입력 이미지보다는 결과 이미지의 화질이 향상된 것을 확인하였습니다.

[그림3 - 'boats_512_high.png']

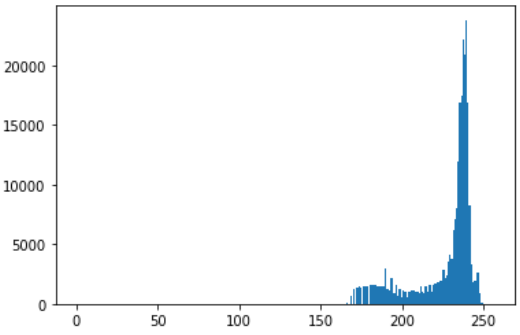
- Python Code

```
inimg = plt.imread('boats_512_high.png') # 위의 코드 중, 이미지 파일명만 바꾼 것입니다.
```

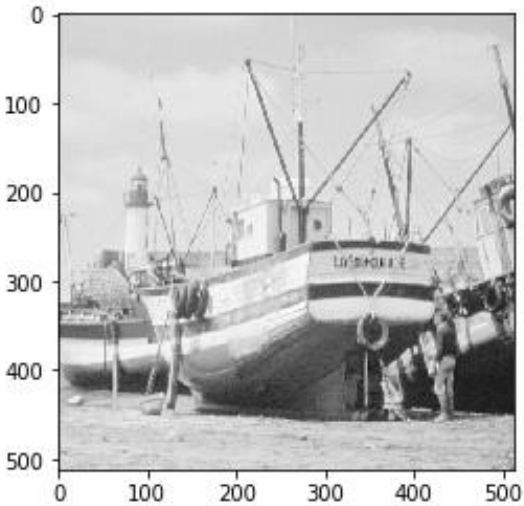
- Image



Input Image



histogram



Output Image

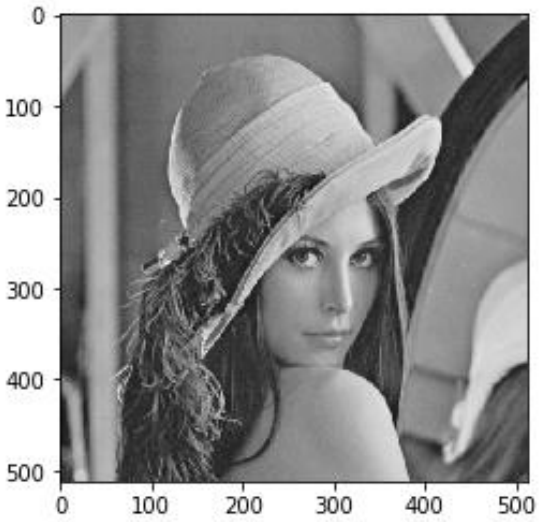
=> Stretching한 결과, 흐릿한 느낌의 입력 이미지에 비해 출력 이미지는 명암대비가 훨씬 높아져 화질이 좋아졌습니다. 위 그림의 입력 이미지는 밝기 값이 높은 영역 즉, 밝은 영역에 영상 픽셀 값이 집중되어 있는데, 이것을 히스토그램 스트레칭 하여 집중되어 있는 것을 퍼트려서 영상의 가시도를 더 좋게 해준 것입니다.

[그림4 - 'lena.png']

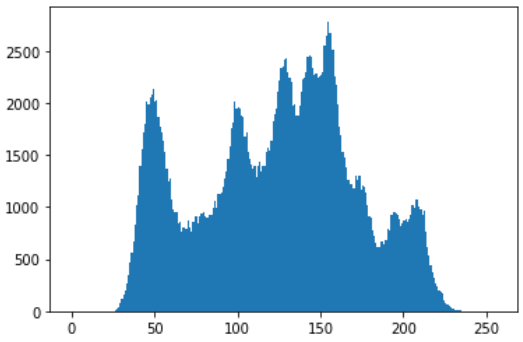
- Python Code

```
inimg = plt.imread('lena.png') # 위의 코드 중, 이미지 파일명만 바꾼 것입니다.
```

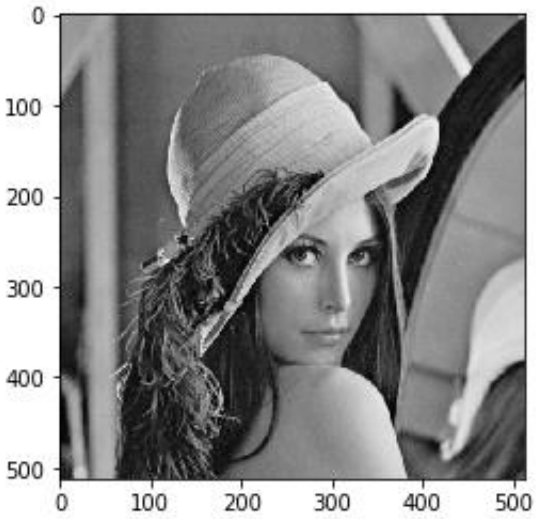
- Image



Input Image



histogram



Output Image

=> Stretching한 결과, 흐릿한 느낌의 입력 이미지에 비해 출력 이미지는 명암대비가 높아져 화질이 더 향상되었습니다. 이미지에 히스토그램 스트레칭 하여 픽셀의 밝기 값들을 퍼트려서 영상의 가시도를 좋게 해준 것입니다.

2-2. Contrast stretching을 하는 cont_st 함수를 작성하고 이 함수를 이용하여 contrast stretching을 하도록 code를 수정하고 2-1과 결과가 같은지 check

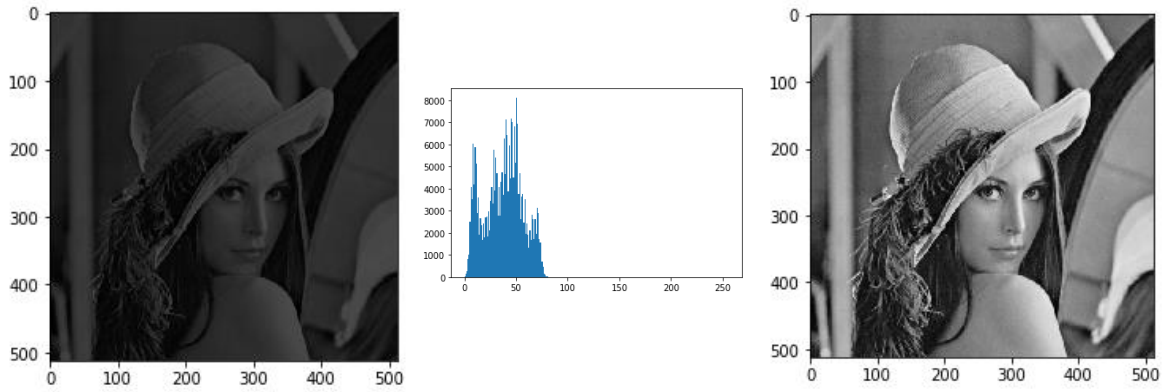
[그림 'lena_512_low1.png' 사용]

- Python Code

```
import scipy
import numpy as np
from scipy import ndimage
from matplotlib import pyplot as plt

# 함수 적용
def cont_st(inimg):
   h,w = inimg.shape
   outimg = np.zeros(inimg.shape)
   maxval = np.amax(inimg)
   minval = np.amin(inimg)
   print(maxval,minval)

   plt.hist(inimg.flatten(),256,[0,256])
   plt.show()
   return outimg

inimg = plt.imread('lena_512_low1.png')
inimg = np.uint8(255*inimg)

plt.imshow(inimg,'gray',vmin=0,vmax=255)
```

```
plt.show()


# cont_st 함수 작성
cont_st = cont_st(inimg)
outimg = np.zeros(inimg.shape)
maxval = np.amax(inimg)
minval = np.amin(inimg)
outimg = 255*np.float32(inimg[:,:]-minval)/float(maxval-minval)
outimg = np.uint8(outimg)
plt.imshow(outimg,'gray',vmin=0,vmax=255)
plt.show()
```

- Image



=> 2-1의 [그림1]과 결과 이미지가 같습니다.

2-3. amax와 amin 함수를 사용하는 대신에 중첩된 for loop을 이용하여 max와 min을 구현하도록 code를 수정하고 2-1과 결과가 같은지 check

[그림 'lena_512_low1.png' 사용]

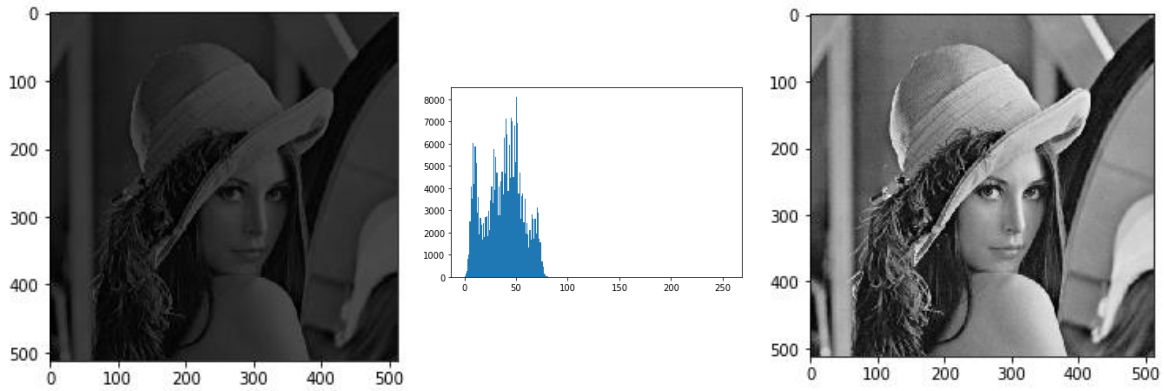
- Python Code

```
import scipy
import numpy as np
```

```
from scipy import ndimage
from matplotlib import pyplot as plt

# 함수 적용
def cont_st(inimg):
   h,w = inimg.shape
   outimg = np.zeros([h,w])
   minval=255      # 이렇게 최대 최소값을 임의로 정하는게 아님.. 최대가 255도 아니구
   maxval=0
   for i in range(h):
       for j in range(w):
           if (inimg[i,j]<minval):
               outimg[i,j] = 255
           else:
               outimg[i,j] = 0        # 중첩된 for loop 사용

   plt.hist(inimg.flatten(),256,[0,256])
   plt.show()
   return outimg

inimg = plt.imread('lena_512_low1.png')
inimg = np.uint8(255*inimg)

plt.imshow(inimg,'gray',vmin=0,vmax=255)
plt.show()

# cont_st 함수 작성
cont_st = cont_st(inimg)
outimg = np.zeros(inimg.shape)
maxval = np.amax(inimg)  # 심지어 amax랑 amin 쓰지말랬는데 썼어^^
minval = np.amin(inimg)
```

```
outimg = 255*np.float32(inimg[:,:]-minval)/float(maxval-minval)
outimg = np.uint8(outimg)
plt.imshow(outimg,'gray',vmin=0,vmax=255)
plt.show()
```

- Image



=> 2-1의 [그림1]과 결과 이미지가 같습니다.

[정답]

```
import numpy as np
from matplotlib import pyplot as plt

# original image & histogram
inimg = plt.imread('lena_512_low1.png')
inimg = np.uint8(255*inimg)

plt.imshow(inimg,'gray',vmin=0,vmax=255)
plt.show()

plt.hist(inimg.flatten(),256,[0,256])
plt.show()

# contrast stretching
h,w = inimg.shape
```

```
maxval = inimg[0,0]  # 최대와 최솟값의 초기값을 지정해준다.
minval = inimg[0,0]
for i in range(h):            # 중첩된 for loop 사용
   for j in range(w):
      if (inimg[i,j] > maxval):
          maxval = inimg[i,j]
      if (inimg[i,j] < minval):
          minval = inimg[i,j]
print(maxval, minval)

# stretched image & histogram
outimg = np.zeros(inimg.shape)
outimg = 255*np.float32(inimg[:,:]-minval)/float(maxval-minval)
outimg = np.uint8(outimg)

plt.imshow(outimg,'gray',vmin=0,vmax=255)
plt.show()

plt.hist(outimg.flatten(),256,[0,256])
plt.show()
```